hp AlphaServer SC

Best Practices I/O Guide

January 2003

This document describes how to administer best practices for I/O on an AlphaServer SC system from the Hewlett-Packard Company.

Revision/Update Information	This is a new manual.
Operating System and Version:	Compaq Tru64 UNIX Version 5.1A, Patch Kit 2
Software Version:	Version 2.5
Maximum Node Count:	1024 nodes
Node Type:	HP AlphaServer ES45 HP AlphaServer ES40 HP AlphaServer DS20L

Legal Notices

The information in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

HEWLETT-PACKARD COMPANY
3000 Hanover Street
Palo Alto, California 94304 U.S.A.

Use of this manual and media is restricted to this product only. Additional copies of the programs may be made for security and back-up purposes only. Resale of the programs, in their present form or with alterations, is expressly prohibited.

Copyright Notices

© 2002 Hewlett-Packard Company
Compaq Computer Corporation is a wholly-owned subsidiary of the Hewlett-Packard Company.

Some information in this document is based on Platform documentation, which includes the following copyright notice:
Copyright 2002 Platform Computing Corporation.

The HP MPI software that is included in this HP AlphaServer SC software release is based on the MPICH V1.2.1 implementation of MPI, which includes the following copyright notice:

© 1993 University of Chicago
© 1993 Mississippi State University

Permission is hereby granted to use, reproduce, prepare derivative works, and to redistribute to others. This software was authored by:

Argonne National Laboratory Group
W. Gropp: (630) 252-4318; FAX: (630) 252-7852; e-mail: gropp@mcs.anl.gov
E. Lusk: (630) 252-5986; FAX: (630) 252-7852; e-mail: lusk@mcs.anl.gov
Mathematics and Computer Science Division, Argonne National Laboratory, Argonne IL 60439

Mississippi State Group
N. Doss and A. Skjellum: (601) 325-8435; FAX: (601) 325-8997; e-mail: tony@erc.msstate.edu
Mississippi State University, Computer Science Department & NSF Engineering Research Center for Computational Field Simulation, P.O. Box 6176, Mississippi State MS 39762

GOVERNMENT LICENSE

Portions of this material resulted from work developed under a U.S. Government Contract and are subject to the following license: the Government is granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license in this computer software to reproduce, prepare derivative works, and perform publicly and display publicly.

DISCLAIMER

This computer code material was prepared, in part, as an account of work sponsored by an agency of the United States Government. Neither the United States, nor the University of Chicago, nor Mississippi State University, nor any of their employees, makes any warranty express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

Trademark Notices

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Expect is public domain software, produced for research purposes by Don Libes of the National Institute of Standards and Technology, an agency of the U.S. Department of Commerce Technology Administration.

Tcl (Tool command language) is a freely distributable language, designed and implemented by Dr. John Ousterhout of Scriptics Corporation.

The following product names refer to specific versions of products developed by Quadrics Supercomputers World Limited ("Quadrics"). These products combined with technologies from HP form an integral part of the supercomputing systems produced by HP and Quadrics. These products have been licensed by Quadrics to HP for inclusion in HP AlphaServer SC systems.

- *Interconnect hardware developed by Quadrics, including switches and adapter cards*
- *Elan, which describes the PCI host adapter for use with the interconnect technology developed by Quadrics*
- *PFS or Parallel File System*
- *RMS or Resource Management System*

Contents

Preface xi

1 hp AlphaServer SC System Overview

1.1 SC System Overview 1–1
1.2 CFS Domains 1–2
1.3 Cluster File System (CFS) 1–3
1.4 Parallel File System (PFS) 1–5
1.5 SC File System (SCFS) 1–5

2 Overview of File Systems and Storage

2.1 Introduction 2–2
2.2 SCFS 2–2
2.2.1 Selection of FAST Mode 2–3
2.2.2 Getting the Most Out of SCFS 2–4
2.3 PFS 2–4
2.3.1 PFS and SCFS 2–6
2.3.1.1 User Process Operation 2–6
2.3.1.2 System Administrator Operation 2–6
2.4 Preferred File Server Nodes and Failover 2–8
2.5 Storage Overview 2–8
2.5.1 Local or Internal Storage 2–9
2.5.1.1 Using Local Storage for Application I/O 2–10
2.5.2 Global or External Storage 2–10
2.5.2.1 System Storage 2–12
2.5.2.2 Data Storage 2–12

3 Managing the Parallel File System (PFS)

3.1 PFS Overview 3–2
3.1.1 PFS Attributes 3–2
3.1.2 Storage Capacity of a PFS File System 3–4
3.2 Planning a PFS File System to Maximize Performance 3–4
3.3 Using a PFS File System 3–6
3.3.1 Creating PFS Files 3–6
3.3.2 Optimizing a PFS File System 3–7

3.3.3 PFS Ioctl Calls . . . 3–9
3.3.3.1 PFSIO_GETFSID . . . 3–10
3.3.3.2 PFSIO_GETMAP . . . 3–10
3.3.3.3 PFSIO_SETMAP . . . 3–10
3.3.3.4 PFSIO_GETDFLTMAP . . . 3–11
3.3.3.5 PFSIO_SETDFLTMAP . . . 3–11
3.3.3.6 PFSIO_GETFSMAP . . . 3–11
3.3.3.7 PFSIO_GETLOCAL . . . 3–12
3.3.3.8 PFSIO_GETFSLOCAL . . . 3–13

4 Managing the SC File System (SCFS)

4.1 SCFS Overview . . . 4–2
4.2 SCFS Configuration Attributes . . . 4–2
4.3 Tuning SCFS . . . 4–5
4.3.1 Tuning SCFS Kernel Subsystems . . . 4–5
4.3.2 Tuning SCFS Server Operations . . . 4–6
4.3.2.1 SCFS I/O Transfers . . . 4–6
4.3.2.2 SCFS Synchronization Management . . . 4–6
4.3.3 Tuning SCFS Client Operations . . . 4–7
4.3.4 Monitoring SCFS Activity . . . 4–7
4.4 SCFS Failover . . . 4–8
4.4.1 SCFS Failover in the File Server Domain . . . 4–8
4.4.2 Failover on an SCFS Importing Node . . . 4–8
4.4.2.1 Recovering from Failure of an SCFS Importing Node . . . 4–8

5 Recommended File System Layout

5.1 Recommended File System Layout . . . 5–2
5.1.1 Stride Size of the PFS . . . 5–4
5.1.2 Stripe Count of the PFS . . . 5–4
5.1.3 Mount Mode of the SCFS . . . 5–4
5.1.4 Home File Systems and Data File Systems . . . 5–5

6 Streamlining Application I/O Performance

6.1 PFS Performance Tuning . . . 6–1
6.2 FORTRAN . . . 6–4
6.3 C . . . 6–5
6.4 Third Party Applications . . . 6–5

Index

List of Figures

Figure 1–1: CFS Makes File Systems Available to All Cluster Members 1–4
Figure 2–1: Example PFS/SCFS Configuration 2–6
Figure 2–2: HP AlphaServer SC Storage Configuration 2–9
Figure 3–1: Parallel File System 3–2

List of Tables

Table 0–1: Abbreviations xiii
Table 0–2: Documentation Conventions xviii
Table 1–1: Node and Member Numbering in an HP AlphaServer SC System 1–2
Table 4–1: SCFS Mount Status Values 4–4

Preface

Purpose of this Guide

This document describes how to administer best practices for I/O on an AlphaServer SC system from the Hewlett-Packard Company ("HP").

Intended Audience

This document is for those who maintain HP AlphaServer SC systems. Some sections will be helpful to end-users, other sections will have information for application engineers, system administrators, system architects and site directors who may be concerned about I/O on an AlphaServer SC system.

Instructions in this document assume that you are an experienced UNIX® administrator who can configure and maintain hardware, operating systems, and networks.

New and Changed Features

This is a new manual so all sections are new.

Structure of This Guide

This document is organized as follows:

- Chapter 1: hp AlphaServer SC System Overview
- Chapter 2: Overview of File Systems and Storage
- Chapter 3: Managing the Parallel File System (PFS)
- Chapter 4: Managing the SC File System (SCFS)
- Chapter 5: Recommended File System Layout
- Chapter 6: Streamlining Application I/O Performance

Related Documentation

You should have a hard copy or soft copy of the following documents:

- *HP AlphaServer SC Release Notes*
- *HP AlphaServer SC Installation Guide*
- *HP AlphaServer SC System Administration Guide*
- *HP AlphaServer SC Interconnect Installation and Diagnostics Manual*
- *HP AlphaServer SC RMS Reference Manual*
- *HP AlphaServer SC User Guide*
- *HP AlphaServer SC Platform LSF® Administrator's Guide*
- *HP AlphaServer SC Platform LSF® Reference Guide*
- *HP AlphaServer SC Platform LSF® User's Guide*
- *HP AlphaServer SC Platform LSF® Quick Reference*
- HP *AlphaServer ES45 Owner's Guide*
- HP *AlphaServer ES40 Owner's Guide*
- HP *AlphaServer DS20L User's Guide*
- HP *StorageWorks HSG80 Array Controller CLI Reference Guide*
- HP *StorageWorks HSG80 Array Controller Configuration Guide*
- HP *StorageWorks Fibre Channel Storage Switch User's Guide*
- HP *StorageWorks Enterprise Virtual Array HSV Controller User Guide*
- HP *StorageWorks Enterprise Virtual Array Initial Setup User Guide*
- HP *SANworks Release Notes - Tru64 UNIX Kit for Enterprise Virtual Array*
- HP *SANworks Installation and Configuration Guide - Tru64 UNIX Kit for Enterprise Virtual Array*
- HP *SANworks Scripting Utility for Enterprise Virtual Array Reference Guide*
- *Compaq TruCluster Server Cluster Release Notes*
- *Compaq TruCluster Server Cluster Technical Overview*
- *Compaq TruCluster Server Cluster Administration*
- *Compaq TruCluster Server Cluster Hardware Configuration*

- *Compaq TruCluster Server Cluster Highly Available Applications*
- *Compaq Tru64 UNIX Release Notes*
- *Compaq Tru64 UNIX Installation Guide*
- *Compaq Tru64 UNIX Network Administration: Connections*
- *Compaq Tru64 UNIX Network Administration: Services*
- *Compaq Tru64 UNIX System Administration*
- *Compaq Tru64 UNIX System Configuration and Tuning*
- *Summit Hardware Installation Guide* from Extreme Networks, Inc.
- *ExtremeWare Software User Guide* from Extreme Networks, Inc.

Note:

The Compaq TruCluster Server documentation set provides a wealth of information about clusters, but there are differences between HP AlphaServer SC clusters and TruCluster Server clusters, as described in the *HP AlphaServer SC System Administration Guide*. You should use the TruCluster Server documentation set to supplement the HP AlphaServer SC documentation set — if there is a conflict of information, use the instructions provided in the HP AlphaServer SC document.

Abbreviations

Table 0–1 lists the abbreviations that are used in this document.

Table 0–1 Abbreviations

Abbreviation	Description
ACL	Access Control List
AdvFS	Advanced File System
API	Application Programming Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
AUI	Attachment Unit Interface
BIND	Berkeley Internet Name Domain
CAA	Cluster Application Availability

Table 0–1 Abbreviations

Abbreviation	Description
CD-ROM	Compact Disc — Read-Only Memory
CDE	Common Desktop Environment
CDFS	CD-ROM File System
CDSL	Context-Dependent Symbolic Link
CFS	Cluster File System
CLI	Command Line Interface
CMF	Console Management Facility
CPU	Central Processing Unit
CS	Compute-Serving
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DMS	Dataless Management Services
DNS	Domain Name System
DRD	Device Request Dispatcher
DRL	Dirty Region Logging
DRM	Distributed Resource Management
EEPROM	Electrically Erasable Programmable Read-Only Memory
ELM	Elan License Manager
EVM	Event Manager
FastFD	Fast, Full Duplex
FC	Fibre Channel
FDDI	Fiber-optic Digital Data Interface
FRU	Field Replaceable Unit
FS	File-Serving
GUI	Graphical User Interface
HBA	Host Bus Adapter

Table 0–1 Abbreviations

Abbreviation	Description
HiPPI	High-Performance Parallel Interface
HPSS	High-Performance Storage System
HWID	Hardware (component) Identifier
ICMP	Internet Control Message Protocol
ICS	Internode Communications Service
IP	Internet Protocol
JBOD	Just a Bunch of Disks
JTAG	Joint Test Action Group
KVM	Keyboard-Video-Mouse
LAN	Local Area Network
LIM	Load Information Manager
LMF	License Management Facility
LSF	Load Sharing Facility
LSM	Logical Storage Manager
MAU	Multiple Access Unit
MB3	Mouse Button 3
MFS	Memory File System
MIB	Management Information Base
MPI	Message Passing Interface
MTS	Message Transport System
NFS	Network File System
NIFF	Network Interface Failure Finder
NIS	Network Information Service
NTP	Network Time Protocol
NVRAM	Non-Volatile Random Access Memory
OCP	Operator Control Panel

Table 0–1 Abbreviations

Abbreviation	Description
OS	Operating System
OSPF	Open Shortest Path First
PAK	Product Authorization Key
PBS	Portable Batch System
PCMCIA	Personal Computer Memory Card International Association
PE	Process Element
PFS	Parallel File System
PID	Process Identifier
PPID	Parent Process Identifier
RAID	Redundant Array of Independent Disks
RCM	Remote Console Monitor
RIP	Routing Information Protocol
RIS	Remote Installation Services
RLA	LSF Adapter for RMS
RMC	Remote Management Console
RMS	Resource Management System
RPM	Revolutions Per Minute
SC	SuperComputer
SCFS	HP AlphaServer SC File System
SCSI	Small Computer System Interface
SMP	Symmetric Multiprocessing
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SRM	System Resources Manager
SROM	Serial Read-Only Memory
SSH	Secure Shell

Table 0–1 Abbreviations

Abbreviation	Description
TCL	Tool Command Language
UBC	Universal Buffer Cache
UDP	User Datagram Protocol
UFS	UNIX File System
UID	User Identifier
UTP	Unshielded Twisted Pair
UUCP	UNIX-to-UNIX Copy Program
WEBES	Web-Based Enterprise Service
WUI	Web User Interface

Documentation Conventions

Table 0–2 lists the documentation conventions that are used in this document.

Table 0–2 Documentation Conventions

Convention	Description
%	A percent sign represents the C shell system prompt.
$	A dollar sign represents the system prompt for the Bourne and Korn shells.
#	A number sign represents the superuser prompt.
P00>>>	A P00>>> sign represents the SRM console prompt.
`Monospace type`	Monospace type indicates file names, commands, system output, and user input.
Boldface type	Boldface type in interactive examples indicates typed user input. Boldface type in body text indicates the first occurrence of a new term.
Italic type	Italic (slanted) type indicates emphasis, variable values, placeholders, menu options, function argument names, and complete titles of documents.
UPPERCASE TYPE	Uppercase type indicates variable names and RAID controller commands.
Underlined type	Underlined type emphasizes important information.
[|] {|}	In syntax definitions, brackets indicate items that are optional and braces indicate items that are required. Vertical bars separating items inside brackets or braces indicate that you choose one item from among those listed.
...	In syntax definitions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.
⋮	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
cat(1)	A cross-reference to a reference page includes the appropriate section number in parentheses. For example, `cat(1)` indicates that you can find information on the `cat` command in Section 1 of the reference pages.
Ctrl/x	This symbol indicates that you hold down the first named key while pressing the key or mouse button that follows the slash.
Note	A note contains information that is of special importance to the reader.
`atlas`	`atlas` is an example system name.

Multiple CFS Domains

The example system described in this document is a 1024-node system, with 32 nodes in each of 32 Cluster File System (CFS) domains. Therefore, the first node in each CFS domain is Node 0, Node 32, Node 64, Node 96, and so on. To set up a different configuration, substitute the appropriate node name(s) for Node 32, Node 64, and so on in this manual.

For information about the CFS domain types supported in HP AlphaServer SC Version 2.5, see Chapter 1.

Location of Code Examples

Code examples are located in the `/Examples` directory of the *HP AlphaServer SC System Software* CD-ROM.

Location of Online Documentation

Online documentation is located in the `/docs` directory of the *HP AlphaServer SC System Software* CD-ROM.

Comments on this Document

HP welcomes any comments and suggestions that you have on this document. Please send all comments and suggestions to your HP Customer Support representative.

1

hp AlphaServer SC System Overview

This guide does not attempt to cover all aspects of normal HP AlphaServer SC system administration (these are covered in detail in the *HP AlphaServer SC System Administration Guide*), but rather focuses on aspects that are specific to the I/O performance.

This chapter is organized as follows:

- SC System Overview (see Section 1.1 on page 1–1)
- CFS Domains (see Section 1.2 on page 1–2)
- Cluster File System (CFS) (see Section 1.3 on page 1–3)
- Parallel File System (PFS) (see Section 1.4 on page 1–5)
- SC File System (SCFS) (see Section 1.5 on page 1–5)

1.1 SC System Overview

An HP AlphaServer SC system is a scalable, distributed-memory, parallel computer system that can expand to up to 4096 CPUs. An HP AlphaServer SC system can be used as a single compute platform to host parallel jobs that consume up to the total compute capacity.

The HP AlphaServer SC system is constructed through the tight coupling of up to 1024 HP AlphaServer ES45 nodes, or up to 128 HP AlphaServer ES40 or HP AlphaServer DS20L nodes. The nodes are interconnected using a high-bandwidth (340 MB/s), low-latency (~3 μs) switched fabric (this fabric is called a **rail**).

For ease of management, the HP AlphaServer SC nodes are organized into multiple Cluster File System (CFS) domains. Each CFS domain shares a common domain file system. This is served by the system storage and provides a common image of the operating system (OS) files to all nodes within a domain. Each node has a locally attached disk, which is used to hold the per-node boot image, swap space, and other temporary files.

1.2 CFS Domains

HP AlphaServer SC Version 2.5 supports multiple Cluster File System (CFS) domains. Each CFS domain can contain up to 32 HP AlphaServer ES45, HP AlphaServer ES40, or HP AlphaServer DS20Ls nodes, providing a maximum of 1024 HP AlphaServer SC nodes.

Nodes are numbered from 0 to 1023 within the overall system, but members are numbered from 1 to 32 within a CFS domain, as shown in Table 1–1, where `atlas` is an example system name.

Table 1–1 Node and Member Numbering in an HP AlphaServer SC System

Node	Member	CFS Domain
atlas0	member1	atlasD0
...	...	
atlas31	member32	
atlas32	member1	atlasD1
...	...	
atlas63	member32	
atlas64	member1	atlasD2
...	...	...
atlas991	member32	
atlas992	member1	atlasD31
...	...	
atlas1023	member32	

System configuration operations must be performed on each of the CFS domains. Therefore, from a system administration point of view, a 1024-node HP AlphaServer SC system may entail managing a single system or managing several CFS domains — this can be contrasted with managing 1024 individual nodes. HP AlphaServer SC Version 2.5 provides several new commands (for example, `scrun`, `scmonmgr`, `scevent`, and `scalertmgr`) that simplify the management of a large HP AlphaServer SC system.

The first two nodes of each CFS domain provide a number of services to the rest of the nodes in their respective CFS domain — the second node also acts as a root file server backup in case the first node fails to operate correctly.

The services provided by the first two nodes of each CFS domain are as follows:

- Serves as the root of the Cluster File System (CFS). The first two nodes in each CFS domain are directly connected to a different Redundant Array of Independent Disks (RAID) subsystem.
- Provides a gateway to an external Local Area Network (LAN). The first two nodes of each CFS domain should be connected to an external LAN.

In HP AlphaServer SC Version 2.5, there are two CFS domain types:

- File-Serving (FS) domain
- Compute-Serving (CS) domain

HP AlphaServer SC Version 2.5 supports a maximum of four FS domains. The SCFS file system exports file systems from an FS domain to the other domains. Although the FS domains can be located anywhere in the HP AlphaServer SC system, HP recommends that you configure either the first domain(s) or the last domain(s) as FS domains — this provides a contiguous range of CS nodes for MPI jobs. It is not mandatory to create an FS domain, but you will not be able to use SCFS if you have not done so. For more information about SCFS, see Chapter 4.

1.3 Cluster File System (CFS)

CFS is a file system that is layered on top of underlying per-node AdvFS file systems. CFS does not change or manage on-disk file system data; rather, it is a value-add layer that provides the following capabilities:

- Shared root file system
  CFS provides each member of the CFS domain with coherent access to all file systems, including the `root` (`/`) file system. All nodes in the file system share the same root.
- Coherent name space
  CFS provides a unifying view of all of the file systems served by the constituent nodes of the CFS domain. All nodes see the same path names. A mount operation by any node is immediately visible to all other nodes. When a node boots into a CFS domain, its file systems are mounted into the domainwide CFS.

  Note:

  One of the nodes physically connected to the root file system storage must be booted first (typically the first or second node of a CFS domain). If another node boots first, it will pause in the boot sequence until the root file server is established.

- High availability and transparent failover
  CFS, in combination with the device request dispatcher, provides disk and file system failover. The loss of a file-serving node does not mean the loss of its served file systems. As long as one other node in the domain has physical connectivity to the relevant storage, CFS will — transparently — migrate the file service to the new node.
- Scalability
  The system is highly scalable, due to the ability to add more active file server nodes.

A key feature of CFS is that every node in the domain is simultaneously a server and a client of the CFS file system. However, this does not mandate a particular operational mode; for example, a specific node can have file systems that are potentially visible to other nodes, but not actively accessed by them. In general, the fact that every node is simultaneously a server and a client is a theoretical point — normally, a subset of nodes will be active servers of file systems into the CFS, while other nodes will primarily act as clients.

Figure 1–1 shows the relationship between file systems contained by disks on a shared SCSI bus and the resulting cluster directory structure. Each member boots from its own boot partition, but then mounts that file system at its mount point in the clusterwide file system. Note that this figure is only an example to show how each cluster member has the same view of file systems in a CFS domain. Many physical configurations are possible, and a real CFS domain would provide additional storage to mirror the critical root (/), /usr, and /var file systems.

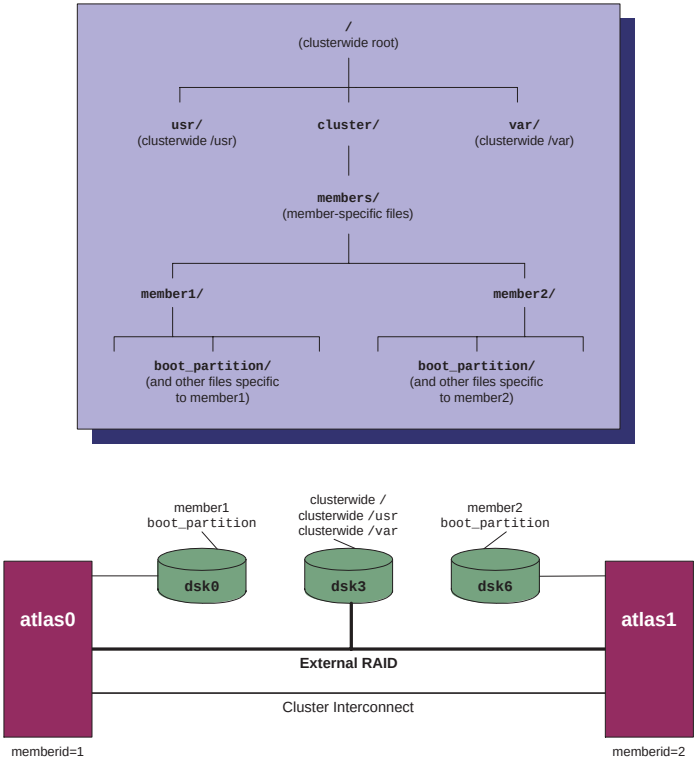

Figure 1–1 CFS Makes File Systems Available to All Cluster Members

See the *HP AlphaServer SC Administration Guide* for more information about the Cluster File System.

1.4 Parallel File System (PFS)

PFS is a higher-level file system, which allows a number of file systems to be accessed and viewed as a single file system view. PFS can be used to provide a parallel application with scalable file system performance. This works by striping the PFS over multiple underlying component file systems, where the component file systems are served by different nodes.

A system does not have to use PFS; where it does, PFS will co-exist with CFS.

See Chapter 3 for more information about PFS.

1.5 SC File System (SCFS)

SCFS provides a global file system for the HP AlphaServer SC system.

The SCFS file system exports file systems from the FS domains to the other domains. It replaces the role of NFS for inter-domain sharing of files within the HP AlphaServer SC system. The SCFS file system is a high-performance system that uses the HP AlphaServer SC Interconnect.

See Chapter 4 for more information about SCFS.

2

Overview of File Systems and Storage

This chapter provides an overview of the file system and storage components of the HP AlphaServer SC system.

The information in this chapter is structured as follows:

- Introduction (see Section 2.1 on page 2–2)
- SCFS (see Section 2.2 on page 2–2)
- PFS (see Section 2.3 on page 2–4)
- Preferred File Server Nodes and Failover (see Section 2.4 on page 2–8)
- Storage Overview (see Section 2.5 on page 2–8)

2.1 Introduction

This section provides an overview of the HP AlphaServer SC Version 2.5 storage and file system capabilities. Subsequent sections provide more detail on administering the specific components.

The HP AlphaServer SC system is comprised of multiple Cluster File System (CFS) domains. There are two types of CFS domains: File-Serving (FS) domains and Compute-Serving (CS) domains. HP AlphaServer SC Version 2.5 supports a maximum of four FS domains.

The nodes in the FS domains serve their file systems, *via* an HP AlphaServer SC high-speed proprietary protocol (SCFS), to the other domains. File system management utilities ensure that the served file systems are mounted at the same point in the name space on all domains.

The result is a data file system (or systems) that is globally visible and performs at high speed. PFS uses the SCFS component file systems to aggregate the performance of multiple file servers, so that users can have access to a single file system with a bandwidth and throughput capability that is greater than a single file server.

2.2 SCFS

With SCFS, a number of nodes in up to four CFS domains are designated as file servers, and these CFS domains are referred to as FS domains. The file server nodes are normally connected to external high-speed storage subsystems (RAID arrays). These nodes serve the associated file systems to the remainder of the system (the other FS domain and the CS domains) *via* the HP AlphaServer SC Interconnect.

Note:

Do not run compute jobs on the FS domains. SCFS I/O is performed by kernel threads that run on the file serving nodes. The kernel threads compete with all other threads on these nodes for I/O bandwidth and CPU availability under the control of the Tru64 UNIX operating system. For this reason, we recommend that you do not run compute jobs on any nodes in the FS domains. Such jobs will compete with the SCFS server threads for machine resources, and so will lower the throughput that the SCFS threads can achieve on behalf of other jobs running on the compute nodes.

The normal default mode of operation for SCFS is to ship data transfer requests directly to the node serving the file system. On the server node, there is a per-file-system SCFS server thread in the kernel. For a write transfer, this thread will transfer the data directly from the user's buffer *via* the HP AlphaServer SC Interconnect and write it to disk.

Data transfers are done in blocks, and disk transfers are scheduled once the block has arrived. This allows large transfers to achieve an overlap between the disk and the HP AlphaServer SC Interconnect. Note that the transfers bypass the client systems' Universal Buffer Cache (UBC). Bypassing the UBC avoids copying data from user space to the kernel prior to shipping it on the network; it allows the system to operate on data sizes larger than the system page size (8KB).

Although bypassing the UBC is efficient for large sequential writes and reads, the data is read by the client multiple times when multiple processes read the same file. While this will still be fast, it is less efficient; therefore, it may be worth setting the mode so that UBC is used (see Section 2.2.1).

2.2.1 Selection of FAST Mode

The default mode of operation for an SCFS file system is set when the system administrator sets up the file system using the `scfsmgr` command (see Chapter 4).

The default mode can be set to FAST (that is, bypasses the UBC) or UBC (that is, uses the UBC). The default mode applies to all files in the file system.

You can override the default mode as follows:

- If the default mode for the file system is UBC, specified files can be used in FAST mode by setting the O_FASTIO option on the file `open()` call.
- If the default mode for the file system is FAST, specified files can be opened in UBC mode by setting the execute bit on the file[1].

Note:

If the default mode is set to UBC, the file system performance and characteristics are equivalent to that expected of an NFS-mounted file system.

1. Note that `mmap()` operations are not supported for FAST files. This is because `mmap()` requires the use of UBC. Executable binaries are normally `mmap`'d by the loader. The exclusion of executable files from the default mode of operation allows binary executables to be used in an SCFS FAST file system.

2.2.2 Getting the Most Out of SCFS

SCFS is designed to deliver high bandwidth transfers for applications performing large serial I/O. Disk transfers are performed by a kernel subsystem on the server node using the HP AlphaServer SC Interconnect kernel-to-kernel message transport. Data is transferred directly from the client process' user space buffer to the server thread without intervening copies.

The HP AlphaServer SC Interconnect reaches its optimum bandwidth at message sizes of 64KB and above. Because of this, optimal SCFS performance will be attained by applications performing transfers that are in excess of this figure. An application performing a single 8MB write is just as efficient as an application performing eight 1MB writes or sixty-four 128KB writes — in fact, a single 8MB write is slightly more efficient, due to the decreased number of system calls.

Because the SCFS system overlaps HP AlphaServer SC Interconnect transfers with storage transfers, optimal user performance will be seen at user transfer sizes of 128KB or greater. Double buffering occurs when a chunk of data (`io_block`, default 128KB) is transferred and is then written to disk while the next 128K is being transferred from the client system *via* the HP AlphaServer SC Elan adapter card.

This allows overlap of HP AlphaServer SC Interconnect transfers and I/O operations. The `sysconfig` parameter `io_block` in the SCFS stanza allows you to tune the amount of data transferred by the SCFS server (see Section 4.3 on page 4–5). The default value is 128KB. If the typical transfer at your site is smaller than 128KB, you can decrease this value to allow double buffering to take effect.

We recommend UBC mode for applications that use short file system transfers — performance will not be optimal if FAST mode is used. This is because FAST mode trades the overhead of mapping the user buffer into the HP AlphaServer SC Interconnect against the efficiency of HP AlphaServer SC Interconnect transfers. Where an application does many short transfers (less than 16KB), this trade-off results in a performance drop. In such cases, UBC mode should be used.

2.3 PFS

Using SCFS, a single FS node can serve a file system or multiple file systems to all of the nodes in the other domains. When normally configured, an FS node will have multiple storage sets (see Section 2.5 on page 2–8), in one of the following configurations:

- There is a file system per storage set — multiple file systems are exported.
- The storage sets are aggregated into a single logical volume using LSM — a single file system is exported.

Where multiple file server nodes are used, multiple file systems will always be exported. This solution can work for installations that wish to scale file system bandwidth by balancing I/O load over multiple file systems. However, it is more generally the case that installations require a single file system, or a small number of file systems, with scalable performance.

PFS provides this capability. A PFS file system is constructed from multiple component file systems. Files in the PFS file system are striped over the underlying component file systems.

When a file is created in a PFS file system, its mapping to component file systems is controlled by a number of parameters, as follows:

- The component file system for the initial stripe

  This is selected at random from the set of components. Using a random selection ensures that the load of multiple concurrent file accesses is distributed.

- The stride size

  This parameter is set at file system creation. It controls how much data is written per file to a component before the next component is used.

- The number of components used in striping

  This parameter is set at file system creation. It specifies the number of components file systems over which an individual file will be striped. The default is all components. In file systems with very large numbers of components, it can be more efficient to use only a subset of components per file (see discussion below).

- The block size

  This number should be less than or equal to the stride size. The stride size must be an even multiple of the block size. The default block size is the same value as the stride size. This parameter specifies how much data the PFS system will issue (in a read or write command) to the underlying file system. Generally, there is not a lot of benefit in changing the default value. SCFS (which is used for the underlying PFS components) is more efficient at bigger transfers, so leaving the block size equal to the stride size maximizes SCFS efficiency.

These parameters are specified at file system creation. They can be modified by a PFS-aware application or library using a set of PFS specific ioctls.

In a configuration with a large number of component file systems and a large client population, it can be more efficient to restrict the number of stripe components. With a large client population writing to every file server, the file servers experience a higher rate of interrupts. By restricting the number of stripe components, individual file server nodes will serve a smaller number of clients, but the aggregate throughput of all servers remains the same. Each client will still get a degree of parallel I/O activity, due to its file being striped

over a number of components. This is true where each client is writing to a different file. If each client process is writing to the same file, it is obviously optimal to stripe over all components.

2.3.1 PFS and SCFS

PFS is a layered file system. It reads and writes data by striping it over component file systems. SCFS is used to serve the component file systems to the CS nodes. Figure 2–1 shows a system with a single FS domain comprised of four nodes, and two CS domains identified as single clients. The FS domain serves the component file systems to the CS domains. A single PFS is built from the component file systems.

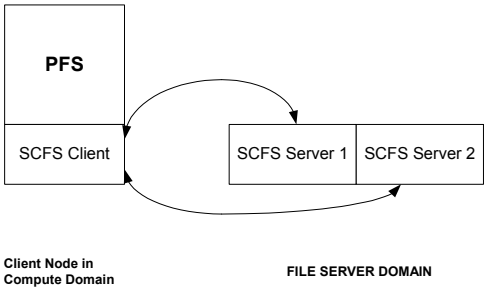

Figure 2–1 Example PFS/SCFS Configuration

2.3.1.1 User Process Operation

Processes running in either (or both) of the CS domains act on files in the PFS system. Depending on the offset within the file, PFS will map the transaction onto one of the underlying SCFS components and pass the call down to SCFS. The SCFS client code passes the I/O request, this time for the SCFS file system, *via* the HP AlphaServer SC Interconnect to the appropriate file server node. At this node, the SCFS thread will transfer the data between the client's buffer and the file system. Multiple processes can be active on the PFS file system at the same time, and can be served by different file server nodes.

2.3.1.2 System Administrator Operation

The file systems in an FS domain are created using the `scfsmgr` command. This command allows the system administrator to specify all of the parameters needed to create and export the file system. The `scfsmgr` command performs the following tasks:

- Creates the AdvFS file domain and file set
- Creates the mount point
- Populates the requisite configuration information in the `sc_scfs` table in the SC database, and in the `/etc/exports` file
- Nominates the preferred file server node

- Synchronizes the other domains, causing the file systems to be imported and mounted at the same mount point

To create the PFS file system, the system administrator uses the `pfsmgr` command to specify the operational parameters for the PFS and identify the component file systems. The `pfsmgr` command performs the following tasks:

- Builds the PFS by creating on-disk data structures
- Creates the mount point for the PFS
- Synchronizes the client systems
- Populates the requisite configuration information in the `sc_pfs` table in the SC database

The following extract shows example contents from the `sc_scfs` table in the SC database:

```
clu_domain      advfs_domain       fset_name     preferred_server  rw   speed    status     mount_point
-------------------------------------------------------------------------------------------------------
atlasD0         scfs0_domain       scfs0         atlas0            rw   FAST     ONLINE     /scfs0
atlasD0         scfs1_domain       scfs1         atlas1            rw   FAST     ONLINE     /scfs1
atlasD0         scfs2_domain       scfs2         atlas2            rw   FAST     ONLINE     /scfs2
atlasD0         scfs3_domain       scfs3         atlas3            rw   FAST     ONLINE     /scfs3
```

In this example, the system administrator created the four component file systems nominating the respective nodes as the preferred file server (see Section 2.4 on page 2–8). This caused each of the CS domains to import the four file systems and mount them at the same point in their respective name spaces. The PFS file system was built on the FS domain using the four component file systems; the resultant PFS file system was mounted on the FS domain. Each of the CS domains also mounted the PFS at the same mount point.

The end result is that each domain sees the same PFS file system at the same mount point. Client PFS accesses are translated into client SCFS accesses and are served by the appropriate SCFS file server node. The PFS file system can also be accessed within the FS domain. In this case, PFS accesses are translated into CFS accesses.

When building a PFS, the system administrator has the following choice:

- Use the set of complete component file systems; for example:
  `/pfs/comps/fs1`; `/pfs/comps/fs2`; `/pfs/comps/fs3`; `/pfs/comps/fs4`
- Use a set of subdirectories within the component file systems; for example:
  `/pfs/comps/fs1/x`; `/pfs/comps/fs2/x`; `/pfs/comps/fs3/x`; `/pfs/comps/fs4/x`

Using the second method allows the system administrator to create different PFS file systems (for instance, with different operational parameters), using the same set of underlying components. This can be useful for experimentation. For production-oriented PFS file systems, the first method is preferred.

2.4 Preferred File Server Nodes and Failover

In HP AlphaServer SC Version 2.5, you can configure up to four FS domains. Although the FS domains can be located anywhere in the HP AlphaServer SC system, we recommend that you configure either the first domain(s) or the last domain(s) as FS domains — this provides a contiguous range of CS nodes for MPI jobs.

Because file server nodes are part of CFS, any member of an FS domain is capable of serving the file system. When an SCFS file system is being configured, one of the configuration parameters specifies the preferred server node. This should be one of the nodes with a direct physical connection to the storage for the file system.

If the node serving a particular component fails, the service will automatically migrate to another node that has connectivity to the storage.

2.5 Storage Overview

There are two types of storage in an HP AlphaServer SC system:

- Local or Internal Storage (see Section 2.5.1 on page 2–9)
- Global or External Storage (see Section 2.5.2 on page 2–10)

Figure 2–2 shows the HP AlphaServer SC storage configuration.

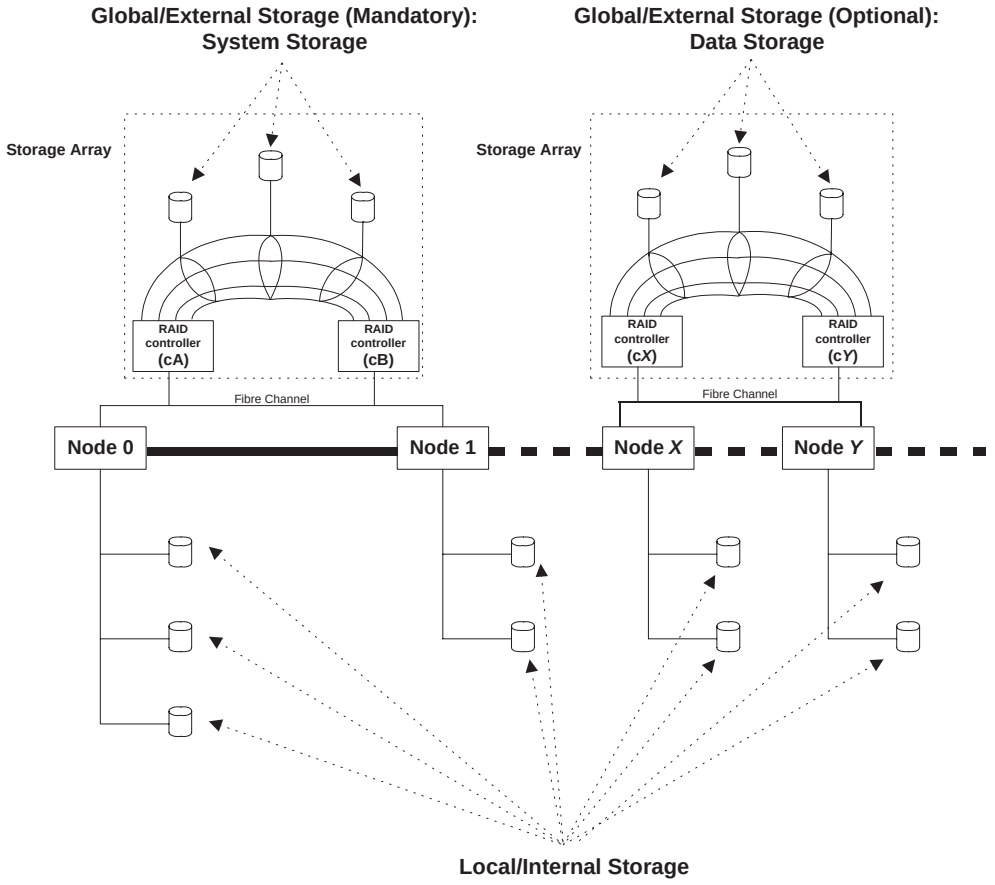

Figure 2–2 HP AlphaServer SC Storage Configuration

2.5.1 Local or Internal Storage

Local or internal storage is provided by disks that are internal to the node cabinet and not RAID-based. Local storage is not highly available. Local disks are intended to store volatile data, not permanent data.

Local storage improves performance by storing copies of node-specific temporary files (for example, swap and core) and frequently used files (for example, the operating system kernel) on locally attached disks.

The SRA utility can automatically regenerate a copy of the operating system and other node-specific files, in the case of disk failure.

Each node requires at least two local disks. The first node of each CFS domain requires a third local disk to hold the base Tru64 UNIX operating system.

The first disk (primary boot disk) on each node is used to hold the following:

- The node's boot partition
- Swap space
- `tmp` and `local` partitions (mounted on `/tmp` and `/local` respectively)
- `cnx h` partition

The second disk (alternate boot disk or backup boot disk) on each node is just a copy of the first disk. In the case of primary disk failure, the system can boot the alternate disk. For more information about the alternate boot disk, see the *HP AlphaServer SC Administration Guide*.

2.5.1.1 Using Local Storage for Application I/O

PFS provides applications with scalable file bandwidth. Some applications have processes that need to write temporary files or data that will be local to that process — for such processes, you can write the temporary data to any local storage that is not used for boot, swap, and core files. If multiple processes in the application are writing data to their own local file system, the available bandwidth is the aggregate of each local file system that is being used.

2.5.2 Global or External Storage

Global or external storage is provided by RAID arrays located in external storage cabinets, connected to a subset of nodes (minimum of two nodes) for availability and throughput.

A HSG-based storage array contains the following in system cabinets with space for disk storage:

- A pair of HSG80 RAID controllers
- Cache modules
- Redundant power supplies

An Enterprise Virtual Array storage system (HSV-based) consists of the following:

- A pair of HSV110 RAID controllers.
- An array of physical disk drives that the controller pair controls. The disk drives are located in drive enclosures that house the support systems for the disk drives.
- Associated physical, electrical, and environmental systems.
- The SANworks HSV Element Manager, which is the graphical interface to the storage system. The element manager software resides on the SANworks Management Appliance and is accessed through a browser.
- SANworks Management Appliance, switches, and cabling.
- At least one host attached through the fabric.

External storage is fully redundant in that each storage array is connected to two RAID controllers, and each RAID controller is connected to at least a pair of host nodes. To provide additional redundancy, a second Fibre Channel switch may be used, but this is not obligatory.

We use the following terms to describe RAID configurations:

- Stripeset (RAID 0)
- Mirrorset (RAID 1)
- RAIDset (RAID 3/5)
- Striped Mirrorset (RAID 0+1)
- JBOD (Just a Bunch Of Disks)

External storage can be organized as Mirrorsets, to ensure that the system continues to function in the event of physical media failure.

External storage is further subdivided as follows:

- System Storage (see Section 2.5.2.1)
- Data Storage (see Section 2.5.2.2)

2.5.2.1 System Storage

System storage is mandatory and is served by the first node in each CFS domain. The second node in each CFS domain is also connected to the system storage, for failover. Node pairs 0 and 1, 32 and 33, 64 and 65, and 96 and 97 each require at least three additional disks, which they will share from the RAID subsystems (Mirrorsets). These disks are required as follows:

- One disk to hold the `/`, `/usr`, and `/var` directories of the CFS domain AdvFS file system
- One disk to be used for generic boot partitions when adding new cluster members
- One disk to be used as a backup during upgrades

Note:

Do not configure a quorum disk in HP AlphaServer SC Version 2.5.

The remaining storage capacity of the external storage subsystem can be configured for user data storage and may be served by any connected node.

System storage must be configured in multiple-bus failover mode.

See Chapter 3 of the *HP AlphaServer SC Installation Guide* for more information on how to configure the external system storage.

2.5.2.2 Data Storage

Data storage is optional and can be served by Node 0, Node 1, and any other nodes that are connected to external storage, as necessary.

See Chapter 3 of the *HP AlphaServer SC Installation Guide* for more information on how to configure the external data storage.

3

Managing the Parallel File System (PFS)

This chapter describes the administrative tasks associated with the Parallel File System (PFS).

The information in this chapter is structured as follows:

- PFS Overview (see Section 3.1 on page 3–2)
- Planning a PFS File System to Maximize Performance (see Section 3.2 on page 3–4)
- Using a PFS File System (see Section 3.3 on page 3–6)

3.1 PFS Overview

A parallel file system (PFS) allows a number of data file systems to be accessed and viewed as a single file system view. The PFS file system stores the data as stripes across the component file systems, as shown in Figure 3–1.

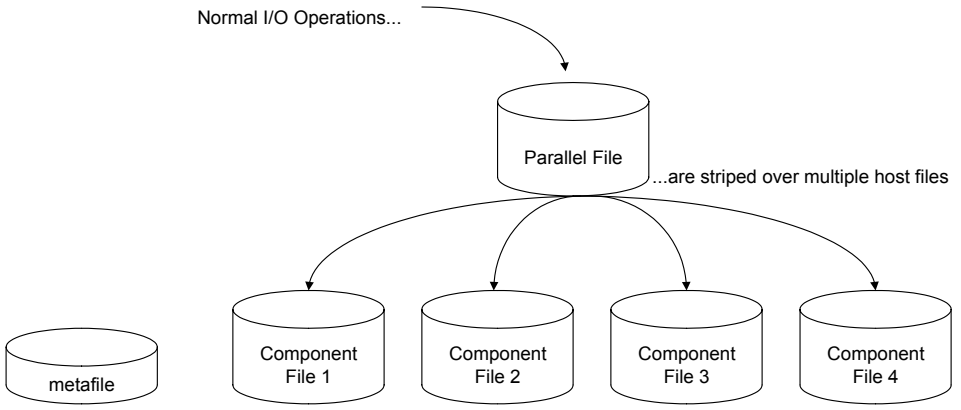

Figure 3–1 Parallel File System

Files written to a PFS file system are written as stripes of data across the set of component file systems. For a very large file, approximately equal portions of a file will be stored on each file system. This can improve data throughput for individual large data read and write operations, because multiple file systems can be active at once, perhaps across multiple hosts.

Similarly, distributed applications can work on large shared datasets with improved performance, if each host works on the portion of the dataset that resides on locally mounted data file systems.

Underlying a component file system is an SCFS file system. The component file systems of a PFS file system can be served by several File-Serving (FS) domains. Where there is only one FS domain, programs running on the FS domain access the component file system *via* the CFS file system mechanisms. Programs running on Compute-Serving (CS) domains access the component file system remotely *via* the SCFS file system mechanisms. If several FS domains are involved in serving components of a PFS file system, each FS domain must import the other domain's SCFS file systems (that is, the SCFS file systems are cross-mounted between domains). See Chapter 4 for a description of FS and CS domains.

3.1.1 PFS Attributes

A PFS file system has a number of attributes, which determine how the PFS striping mechanism operates for files within the PFS file system. Some of the attributes, such as the set of component file systems, can only be configured when the file system is created, so you should plan these carefully (see Section 3.2 on page 3–4). Other attributes, such as the size of the stride, can be reconfigured after file system creation; these attributes can also be configured on a per-file basis.

The PFS attributes are as follows:

- NumFS (Component File System List)

  A PFS file system is comprised of a number of component file systems. The component file system list is configured when a PFS file system is created.

- Block (Block Size)

  The block size is the maximum amount of data that will be processed as part of a single operation on a component file system. The block size is configured when a PFS file system is created.

- Stride (Stride Size)

  The stride size is the amount (or stride) of data that will be read from, or written to, a single component file system before advancing to the next component file system, selected in a round robin fashion. The stride value must be an integral multiple of the block size (see Block above).

  The default stride value is defined when a PFS file system is created, but this default value can be changed using the appropriate ioctl (see Section 3.3.3.5 on page 3–11). The stride value can also be reconfigured on a per-file basis using the appropriate ioctl (see Section 3.3.3.3 on page 3–10).

- Stripe (Stripe Count)

  The stripe count specifies the number of component file systems to stripe data across, in cyclical order, before cycling back to the first file system. The stripe count must be non-zero, and less than or equal to the number of component file systems (see NumFS above).

  The default stripe count is defined when a PFS file system is created, but this default value can be changed using appropriate ioctl (see Section 3.3.3.5 on page 3–11). The stripe count can also be reconfigured on a per-file basis using the appropriate ioctl (see Section 3.3.3.3 on page 3–10).

- Base (Base File System)

  The base file system is the index of the file system, in the list of component file systems, that contains the first stripe of file data. The base file system must be between 0 and NumFS – 1 (see NumFS above).

  The default base file system is selected when the file is created, based on the modulus of the file `inode` number and the number of component file systems. The base file system can also be reconfigured on a per-file basis using the appropriate ioctl (see Section 3.3.3.3 on page 3–10).

3.1.2 Storage Capacity of a PFS File System

The storage capacity of a PFS file system is primarily dependent on the capacity of the component file systems, but also depends on how the individual files are laid out across the component file systems.

For a particular file, the maximum storage capacity available within the PFS file system can be calculated by multiplying the stripe count (that is, the number of file systems it is striped across) by the actual storage capacity of the smallest of these component file systems.

Note:

The PFS file system stores directory mapping information on the first (root) component file system. The PFS file system uses this mapping information to resolve files to their component data file system block. Because of the minor overhead associated with this mapping information, the actual capacity of the PFS file system will be slightly reduced, unless the root component file system is larger than the other component file systems.

For example, a PFS file system consists of four component file systems (A, B, C, and D), with actual capacities of 3GB, 1GB, 3GB, and 4GB respectively. If a file is striped across all four file systems, then the maximum capacity of the PFS for this file is 4GB — that is, 1GB (Minimum Capacity) x 4 (File Systems). However, if a file is only striped across component file systems C and D, then the maximum capacity would be 6GB — that is, 3GB (Minimum Capacity) x 2 (File Systems).

For information on how to extend the storage capacity of PFS file systems, see the *HP AlphaServer SC Administration Guide*.

3.2 Planning a PFS File System to Maximize Performance

The primary goal, when using a PFS file system, is to achieve improved file access performance, scaling linearly with the number of component file systems (NumFS). However, it is possible for more than one component file system to be served by the same server, in which case the performance may only scale linearly with the number of servers.

To achieve this goal, you must analyze the intended use of the PFS file system. For a given application or set of applications, determine the following criteria:

- Number of Files

  An important factor when planning a PFS file system is the expected number of files.

If expecting to use a very large number of files in a large number of directories, then you should allow extra space for PFS file metadata on the first (root) component file system. The extra space required will be similar in size to the overhead required to store the files on an AdvFS file system.

- Access Patterns

  How data files will be accessed, and who will be accessing the files, are two very important criteria when determining how to plan a PFS file system.

  If a file is to be shared among a number of process elements (PEs) on different nodes on the CFS domain, you can improve performance by ensuring that the file layout matches the access patterns, so that all PEs are accessing the parts of a file that are local to their nodes.

  If files are specific to a subset of nodes, then localizing the file to the component file systems that are local to these nodes should improve performance.

  If a large file is being scanned in a sequential or random fashion, then spreading the file over all of the component file systems should benefit performance.

- File Dynamics and Lifetime

  Data files may exist for only a brief period while an application is active, or they may persist across multiple runs. During this time, their size may alter significantly.

  These factors affect how much storage must be allocated to the component file systems, and whether backups are required.

- Bandwidth Requirements

  Applications that run for very long periods of time frequently save internal state at regular intervals, allowing the application to be restarted without losing too much work.

  Saving this state information can be a very I/O intensive operation, the performance of which can be improved by spreading the write over multiple physical file systems using PFS. Careful planning is required to ensure that sufficient I/O bandwidth is available.

To maximize the performance gain, some or all of the following conditions should be met:

1. PFS file systems should be created so that files are spread over the appropriate component file systems or servers. If only a subset of nodes will be accessing a file, then it may be useful to limit the file layout to the subset of component file systems that are local to these nodes, by selecting the appropriate stripe count.
2. The amount of data associated with an operation is important, as this determines what the stride and block sizes should be for a PFS file system. A small block size will require more I/O operations to obtain a given amount of data, but the duration of the operation will be shorter. A small stride size will cycle through the set of component file systems faster, increasing the likelihood of multiple file systems being active simultaneously.

3. The layout of a file should be tailored to match the access pattern for the file. Serial access may benefit from a small stride size, delivering improved read or write bandwidth. Random access performance should improve as more than one file system may seek data at the same time. Strided data access may require careful tuning of the PFS block size and the file data stride size to match the size of the access stride.
4. The base file system for a file should be carefully selected to match application access patterns. In particular, if many files are accessed in lock step, then careful selection of the base file system for each file can ensure that the load is spread evenly across the component file system servers. Similarly, when a file is accessed in a strided fashion, careful selection of the base file system may be required to spread the data stripes appropriately.

3.3 Using a PFS File System

A PFS file system supports POSIX semantics and can be used in the same way as any other Tru64 UNIX file system (for example, UFS or AdvFS), except as follows:

- PFS file systems are mounted with the `nogrpid` option implicitly enabled. Therefore, SVID III semantics apply. For more details, see the AdvFS/UFS options for the `mount(8)` command.
- The layout of the PFS file system, and of files residing on it, can be interrogated and changed using special PFS ioctl calls (see Section 3.3.3 on page 3–9).
- The PFS file system does not support file locking using the `lockf(2)`, `fcntl(2)`, or `lockf(3)` interfaces.
- PFS provides support for the `mmap()` system call for multicomponent file systems, sufficient to allow the execution of binaries located on a PFS file system. This support is, however, not always robust enough to support how some compilers, linkers, and profiling tools make use of the `mmap()` system call when creating and modifying binary executables. Most of these issues can be avoided if the PFS file system is configured to use a stripe count of 1 by default; that is, use only a single data component per file.

The information in this section is organized as follows:

- Creating PFS Files (see Section 3.3.1 on page 3–6)
- Optimizing a PFS File System (see Section 3.3.2 on page 3–7)
- PFS Ioctl Calls (see Section 3.3.3 on page 3–9)

3.3.1 Creating PFS Files

When a user creates a file, it inherits the default layout characteristics for that PFS file system, as follows:

- Stride size — the default value is inherited from the `mkfs_pfs` command.

- Number of component file systems — the default is to use all of the component file systems.
- File system for the initial stripe — the default value for this is chosen at random.

You can override the default layout on a per-file basis using the `PFSIO_SETMAP` ioctl on file creation.

Note:

This will truncate the file, destroying the content. See Section 3.3.3.3 on page 3–10 for more information about the `PFSIO_SETMAP` ioctl.

PFS file systems also have the following characteristics:

- Copying a sequential file to a PFS file system will cause the file to be striped. The stride size, number of component file systems, and start file are all set to the default for that file system.
- Copying a file from a PFS file system to the same PFS file system will reset the layout characteristics of the file to the default values.

3.3.2 Optimizing a PFS File System

The performance of a PFS file system is improved if accesses to the component data on the underlying CFS file systems follow the performance guidelines for CFS. The following guidelines will help to achieve this goal:

1. In general, consider the stripe count of the PFS file system.

   If a PFS is formed from more than 8 component file systems, we recommend setting the default stripe count to a number that is less than the total number of components. This will reduce the overhead incurred when creating and deleting files, and improve the performance of applications that access numerous small-to-medium-sized files.

   For example, if a PFS file system is constructed using 32 components, we recommend selecting a default stripe count of 8 or 4. The desired stripe count for a PFS can be specified when the file system is created, or using the PFSIO_SETDFLTMAP ioctl. See Section 3.3.3.5 on page 3–11 for more information about the PFSIO_SETDFLTMAP ioctl.

2. For PFS file systems consisting of FAST-mounted SCFS components, consider the stride size.

   As SCFS FAST mode is optimized for large I/O transfers, it is important to select a stride size that takes advantage of SCFS while still taking advantage of the parallel I/O capabilities of PFS. We recommend setting the stride size to at least 512K.

   To make efficient use of both PFS and SCFS capabilities, an application should read or write data in sizes that are multiples of the stride size.

For example, a large file is being written to a 32-component PFS, the stripe count for the file is 8, and the stride size is 512K. If the file is written in blocks of 4MB or more, this will make maximum use of both the PFS and SCFS capabilities, as it will generate work for all of the component file systems on every write. However, setting the stride size to 64K and writing in blocks of 512K is not a good idea, as it will not make good use of SCFS capabilities.

3. For PFS file systems consisting of UBC-mounted SCFS components, follow these guidelines:

   - Avoid False Sharing

     Try to lay the file out across the component file systems such that only one node is likely to access a particular stripe of data. This is especially important when writing data.

     False sharing occurs when two nodes try to get exclusive access to different parts of the same file. This causes the nodes to repeatedly seek access to the file, as their privileges are revoked.

   - Maximize Caching Benefits

     A second order effect that can be useful is to ensure that regions of a file are distributed to individual nodes. If one node handles all the operations on a particular region, then the CFS Client cache is more likely to be useful, reducing the network traffic associated with accessing data on remote component file systems.

File system tools, such as backup and restore utilities, can act on the underlying CFS file system without integrating with the PFS file system.

External file managers and movers, such as the High Performance Storage System (HPSS) and the parallel file transfer protocol (pftp), can achieve good parallel performance by accessing PFS files in a sequential (stride = 1) fashion. However, the performance may be further improved by integrating the mover with PFS, so that it understands the layout of a PFS file. This enables the mover to alter its access patterns to match the file layout.

3.3.3 PFS Ioctl Calls

Valid PFS ioctl calls are defined in the `map.h` header file (`<sys/fs/pfs/map.h>`) on an installed system. A PFS ioctl call requires an open file descriptor for a file (either the specific file being queried or updated, or any file) on the PFS file system.

In PFS ioctl calls, the N different component file systems are referred to by index number (`0` to `N-1`). The index number is that of the corresponding symbolic link in the component file system root directory.

The sample program `ioctl_example.c`, provided in the `/Examples/pfs-example` directory on the *HP AlphaServer SC System Software* CD-ROM, demonstrates the use of PFS ioctl calls.

HP AlphaServer SC Version 2.5 supports the following PFS ioctl calls:

- PFSIO_GETFSID (see Section 3.3.3.1 on page 3–10)
- PFSIO_GETMAP (see Section 3.3.3.2 on page 3–10)
- PFSIO_SETMAP (see Section 3.3.3.3 on page 3–10)
- PFSIO_GETDFLTMAP (see Section 3.3.3.4 on page 3–11)
- PFSIO_SETDFLTMAP (see Section 3.3.3.5 on page 3–11)
- PFSIO_GETFSMAP (see Section 3.3.3.6 on page 3–11)
- PFSIO_GETLOCAL (see Section 3.3.3.7 on page 3–12)
- PFSIO_GETFSLOCAL (see Section 3.3.3.8 on page 3–13)

Note:

The following ioctl calls will be supported in a future version of the HP AlphaServer SC system software:

PFSIO_HSMARCHIVE — Instructs PFS to archive the given file.

PFSIO_HSMISARCHIVED — Queries if the given PFS file is archived or not.

3.3.3.1 PFSIO_GETFSID

Description: For a given PFS file, retrieves the ID for the PFS file system.
This is a unique 128-bit value.

Data Type: `pfsid_t`

Example: `376a643c-000ce681-00000000-4553872c`

3.3.3.2 PFSIO_GETMAP

Description: For a given PFS file, retrieves the mapping information that specifies how it is laid out across the component file systems.
This information includes the number of component file systems, the ID of the component file system containing the first data block of a file, and the stride size.

Data Type: `pfsmap_t`

Example: The PFS file system consists of two components, 64KB stride:
Slice: Base = 0 Count = 2
Stride: 65536
This configures the file to be laid out with the first block on the first component file system, and a stride size of 64KB.

3.3.3.3 PFSIO_SETMAP

Description: For a given PFS file, sets the mapping information that specifies how it is laid out across the component file systems. Note that this will truncate the file, destroying the content.
This information includes the number of component file systems, the ID of the component file system containing the first data block of a file, and the stride size.

Data Type: `pfsmap_t`

Example: The PFS file system consists of three components, 64KB stride:
Slice: Base = 2 Count = 3
Stride: 131072
This configures the file to be laid out with the first block on the third component file system, and a stride size of 128KB. (The stride size of the file can be an integral multiple of the PFS block size.)

3.3.3.4 PFSIO_GETDFLTMAP

Description: For a given PFS file system, retrieves the default mapping information that specifies how newly created files will be laidout across the component file systems.
This information includes the number of component file systems, the ID of the component file system containing the first data block of a file, and the stride size.

Data Type: `pfsmap_t`

Example: See `PFSIO_GETMAP` (Section 3.3.3.2 on page 3–10).

3.3.3.5 PFSIO_SETDFLTMAP

Description: For a given PFS file system, sets the default mapping information that specifies how newly created files will be laidout across the component file systems.
This information includes the number of component file systems, the ID of the component file system containing the first data block of a file, and the stride size.

Data Type: `pfsmap_t`

Example: See `PFSIO_SETMAP` (Section 3.3.3.3 on page 3–10).

3.3.3.6 PFSIO_GETFSMAP

Description: For a given PFS file system, retrieves the number of component file systems, and the default stride size.

Data Type: `pfsmap_t`

Example: The PFS file system consists of eight components, 128KB stride:
Slice: Base = 0 Count = 8
Stride: 131072
This configures the file to be laid out with the first block on the first component file system, and a stride size of 128KB. For `PFSIO_GETFSMAP`, the base is always 0 — the component file system layout is always described with respect to a base of 0.

3.3.3.7 PFSIO_GETLOCAL

Description: For a given PFS file, retrieves information that specifies which parts of the file are local to the host.
This information consists of a list of slices, taken from the layout of the file across the component file systems, that are local. Blocks laid out across components that are contiguous are combined into single slices, specifying the block offset of the first of the components, and the number of contiguous components.

Data Type: `pfsslices_ioctl_t`

Example: a) The PFS file system consists of three components, all local, file starts on first component:
Size: 3
Count: 1
Slice: Base = 0 Count = 3
b) The PFS file system consists of three components, second is local, file starts on first component:
Size: 3
Count: 1
Slice: Base = 1 Count = 1
c) The PFS file system consists of three components, second is remote, file starts on first component:
Size: 3
Count: 2
Slices: Base = 0 Count = 1
Base = 2 Count = 1
d) The PFS file system consists of three components, second is remote, file starts on second component:
Size: 3
Count: 1
Slice: Base = 1 Count = 2

3.3.3.8 PFSIO_GETFSLOCAL

Description: For a given PFS file system, retrieves information that specifies which of the components are local to the host.
This information consists of a list of slices, taken from the set of components, that are local. Components that are contiguous are combined into single slices, specifying the ID of the first component, and the number of contiguous components.

Data Type: `pfsslices_ioctl_t`

Example: a) The PFS file system consists of three components, all local:
Size: 3
Count: 1
Slice: Base = 0 Count = 3
b) The PFS file system consists of three components, second is local:
Size: 3
Count: 1
Slice: Base = 1 Count = 1
c) The PFS file system consists of three components, second is remote:
Size: 3
Count: 2
Slices: Base = 0 Count = 1
Base = 2 Count = 1

4

Managing the SC File System (SCFS)

The SC file system (SCFS) provides a global file system for the HP AlphaServer SC system.

The information in this chapter is arranged as follows:

- SCFS Overview (see Section 4.1 on page 4–2)
- SCFS Configuration Attributes (see Section 4.2 on page 4–2)
- Tuning SCFS (see Section 4.3 on page 4–5)
- SCFS Failover (see Section 4.4 on page 4–8)

4.1 SCFS Overview

The HP AlphaServer SC system is comprised of multiple Cluster File System (CFS) domains. There are two types of CFS domains: File-Serving (FS) domains and Compute-Serving (CS) domains. HP AlphaServer SC Version 2.5 supports a maximum of four FS domains.

The SCFS file system exports file systems from an FS domain to the other domains. Therefore, it provides a global file system across all nodes of the HP AlphaServer SC system. The SCFS file system is a high-performance file system that is optimized for large I/O transfers. When accessed *via* the FAST mode, data is transferred between the client and server nodes using the HP AlphaServer SC Interconnect network for efficiency.

SCFS file systems may be configured by using the `scfsmgr` command. You can use the `scfsmgr` command or SysMan Menu, on any node or on a management server (if present), to manage all SCFS file systems. The system automatically reflects all configuration changes on all domains. For example, when you place an SCFS file system on line, it is mounted on all domains.

The underlying storage of an SCFS file system is an AdvFS fileset on an FS domain. Within an FS domain, access to the file system from any node is managed by the CFS file system and has the usual attributes of CFS file systems (common mount point, coherency, and so on). An FS domain serves the SCFS file system to nodes in the other domains. In effect, an FS domain exports the file system, and the other domains import the file system.

This is similar to — and, in fact, uses features of — the NFS system. For example, `/etc/exports` is used for SCFS file systems. The mount point of an SCFS file system uses the same name throughout the HP AlphaServer SC system so there is a coherent file name space. Coherency issues related to data and metadata are discussed later.

4.2 SCFS Configuration Attributes

The SC database contains SCFS configuration data. The `/etc/fstab` file is not used to manage the mounting of SCFS file systems. However, the `/etc/exports` is used for this purpose. Use SysMan Menu or the `scfsmgr` command to edit this configuration data — do not update the contents of the SC database directly. Do not add entries to, or remove entries from, the `/etc/exports` file. Once entries have been created, you can edit the `/etc/exports` file in the usual way.

An SCFS file system is described by the following attributes:

- AdvFS domain and fileset name

  This is the name of the AdvFS domain and fileset that contains the underlying data storage of an SCFS file system. This information is only used by the FS domain that serves the SCFS file system. However, although AdvFS domain and fileset names generally need only be unique within a given CFS domain, the SCFS system uses unique names. Therefore, the AdvFS domain and fileset name must be unique across the HP AlphaServer SC system.

  In addition, HP recommends the following conventions:

  – You should use only one AdvFS fileset in an AdvFS domain.

  – The domain and fileset names should use a common root name. For example, an appropriate name would be `data_domain#data`.

  SysMan Menu uses these conventions. The `scfsmgr` command allows more flexibility.

- Mountpoint

  This is the pathname of the mountpoint for the SCFS file system. This is the same on all CFS domains in the HP AlphaServer SC system.

- Preferred Server

  This specifies the node that normally serves the file system. When an FS domain is booted, the first node that has access to the storage will mount the file system. When the preferred server boots, it takes over the serving of that storage. For best performance, the preferred server should have direct access to the storage. The `cfsmgr` command controls which node serves the storage.

- Read/Write or Read-Only

  This has exactly the same syntax and meaning as in an NFS file system.

- FAST or UBC

  This attribute refers to the default behavior of clients accessing the FS domain. The client has two possible paths to access the FS domain:

  – Bypass the Universal Buffer Cache (UBC) and access the serving node directly. This corresponds to the FAST mode.

    The FAST mode is suited to large data transfers where bypassing the UBC provides better performance. In addition, since accesses are made directly to the serving node, multiple writes by several client nodes are serialized; hence, data coherency is preserved. Multiple readers of the same data will all have to obtain the data individually from the server node since the UBC is bypassed on the client nodes.

    While a file is opened *via* the FAST mode, all subsequent file `open()` calls on that cluster will inherit the FAST attribute even if not explicitly specified.

– Access is through the UBC. This corresponds to the UBC mode.

  The UBC mode is suited to small data transfers, such as those produced by formatted writes in Fortran. Data coherency has the same characteristics as NFS.

  If a file is currently opened *via* the UBC mode, and a user attempts to open the same file *via* the FAST mode, an error (EINVAL) is returned to the user.

Whether the SCFS file system is mounted FAST or UBC, the access for individual files is overridden as follows:

– If the file has an executable bit set, access is *via* the UBC; that is, uses the UBC path.

– If the file is opened with the O_SCFSIO option (defined in <`sys/scfs.h`>), access is *via* the FAST path.

- ONLINE or OFFLINE

  You do not directly mount or unmount SCFS file systems. Instead, you mark the SCFS file system as ONLINE or OFFLINE. When you mark an SCFS file system as ONLINE, the system will mount the SCFS file system on all CFS domains. When you mark the SCFS file system as OFFLINE, the system will unmount the file system on all CFS domains.

  The state is persistent. For example, if an SCFS file system is marked ONLINE and the system is shut down and then rebooted, the SCFS file system will be mounted as soon as the system has completed booting.

- Mount Status

  This indicates whether an SCFS file system is mounted or not. This attribute is specific to a CFS domain (that is, each CFS domain has a mount status). The mount status values are listed in Table 4–1.

Table 4–1 SCFS Mount Status Values

Mount Status	Description
`mounted`	The SCFS file system is mounted on the domain.
`not-mounted`	The SCFS file system is not mounted on the domain.
`mounted-busy`	The SCFS file system is mounted, but an attempt to unmount it has failed because the SCFS file system is in use. When a PFS file system uses an SCFS file system as a component of the PFS, the SCFS file system is in use and cannot be unmounted until the PFS file system is also unmounted. In addition, if a CS domain fails to unmount the SCFS, the FS domain does not attempt to unmount the SCFS, but instead marks it as `mounted-busy`.

Table 4–1 SCFS Mount Status Values

Mount Status	Description
`mounted-stale`	The SCFS file system is mounted, but the FS domain that serves the file system is no longer serving it. Generally, this is because the FS domain has been rebooted — for a period of time, the CS domain sees `mounted-stale` until the FS domain has finished mounting the AdvFS file systems underlying the SCFS file system. The `mounted-stale` status only applies to CS domains.
`mount-not-served`	The SCFS file system was mounted, but all nodes of the FS domain that can serve the underlying AdvFS domain have left the domain.
`mount-failed`	An attempt was made to mount the file system on the domain, but the mount command failed. When a mount fails, the reason for the failure is reported as an event of class `scfs` and type `mount.failed`. See *HP AlphaServer SC Administration Guide* for details on how to access this event type.
`mount-noresponse`	The file system is mounted; however, the FS domain is not responding to client requests. Usually, this is because the FS domain is shut down.
`mounted-io-err`	The file system is mounted, but when you attempt to access it, programs get an I/O Error. This can happen on a CS domain when the file system is in the `mount-not-served` state on the FS domain.
`unknown`	Usually, this indicates that the FS domain or CS domain is shut down. However, a failure of an FS or CS domain to respond can also cause this state.

The attributes of SCFS file systems can be viewed using the `scfsmgr show` command.

4.3 Tuning SCFS

The information in this section is organized as follows:

- Tuning SCFS Kernel Subsystems (see Section 4.3.1 on page 4–5)
- Tuning SCFS Server Operations (see Section 4.3.2 on page 4–6)
- Tuning SCFS Client Operations (see Section 4.3.3 on page 4–7)
- Monitoring SCFS Activity (see Section 4.3.4 on page 4–7)

4.3.1 Tuning SCFS Kernel Subsystems

To tune any of the SCFS subsystem attributes permanently, you must add an entry to the appropriate subsystem stanza, either `scfs` or `scfs_client`, in the `/etc/sysconfigtab` file. Do not edit the `/etc/sysconfigtab` file directly — use the `sysconfigdb` command to view and update its contents. Changes made to the `/etc/sysconfigtab` file will take

effect when the system is next booted. Some of the attributes can also be changed dynamically using the `sysconfig` command, but these settings will be lost after a reboot unless the changes are also added to the `/etc/sysconfigtab` file.

4.3.2 Tuning SCFS Server Operations

A number of configurable attributes in the `scfs` kernel subsystem affect SCFS serving. Some of these attributes can be dynamically configured, while others require a reboot before they take effect. For a detailed explanation of the `scfs` subsystem attributes, see the `sys_attrs_scfs(5)` reference page.

The default settings for the `scfs` subsystem attributes should work well for a mixed work load. However, performance may be improved by tuning some of the parameters.

4.3.2.1 SCFS I/O Transfers

SCFS I/O achieves best performance results when processing large I/O requests.

If a client generates a very large I/O request, such as writing 512MB of data to a file, this request will be performed as a number of smaller operations. The size of these smaller operations is dictated by the `io_size` attribute of the server node for the SCFS file system. The default value of the `io_size` attribute is 16MB.

This subrequest is then sent to the SCFS server, which in turn performs the request as a number of smaller operation. This time, the size of the smaller operations is specified by the `io_block` attribute. The default value of the `io_block` attribute is 128KB. This allows the SCFS server to implement a simple double-buffering scheme which overlaps I/O and interconnect transfers.

Performance for very large requests may be improved by increasing the `io_size` attribute, though this will increase the setup time for each request on the client. You must propagate this change to every node in the FS domain, and then reboot the FS domain.

Performance for smaller transfers (<256KB) may also be improved slightly by reducing the `io_block` size, to increase the effect of the double-buffering scheme. You must propagate this change to every node in the FS domain, and then reboot the FS domain.

4.3.2.2 SCFS Synchronization Management

The SCFS server will synchronize the dirty data associated with a file to disk, if one or more of the following criteria is true:

- The file has been dirty for longer than `sync_period` seconds. The default value of the `sync_period` attribute is 10.
- The amount of dirty data associated with the file exceeds `sync_dirty_size`. The default value of the `sync_dirty_size` attribute is 64MB.

- The number of write transactions since the last synchronization exceeds `sync_handle_trans`. The default value of the `sync_handle_trans` attribute is 204.

If an application generates a workload that causes one of these conditions to be reached very quickly, poor performance may result because I/O to a file regularly stalls waiting for the synchronize operation to complete. For example, if an application writes data in 128KB blocks, the default `sync_handle_trans` value would be exceeded after writing 25.5MB. Performance may be improved by increasing the `sync_handle_trans` value. You must propagate this change to every node in the FS domain, and then reboot the FS domain.

Conversely, an application may generate a workload that does not cause the `sync_dirty_size` and `sync_handle_trans` limits to be exceeded — for example, an application that writes 32MB in large blocks to a number of different files. In such cases, the data is not synchronized to disk until the `sync_period` has expired. This could result in poor performance as UBC resources are rapidly consumed, and the storage subsystems are left idle. Tuning the dynamically reconfigurable attribute `sync_period` to a lower value may improve performance in this case.

4.3.3 Tuning SCFS Client Operations

The `scfs_client` kernel subsystem has one configurable attribute.The `max_buf` attribute specifies the maximum amount of data that a client will allow to be shadow-copied for an SCFS file system, before blocking new requests from being issued. The default value of the `max_buf` attribute is 256MB, and can be dynamically modified.

The client keeps shadow copies of data written to an SCFS file system so that, in the event of a server crash, the requests can be re-issued.

The SCFS server notifies clients when requests have been synchronized to disk so that they can release the shadow copies, and allow new requests to be issued.

If a client node is accessing many SCFS file systems, for example, *via* a PFS file system (see Chapter 3), it may be better to reduce the `max_buf` setting. This will minimize the impact of maintaining many shadow copies for the data written to the different file systems.

For a detailed explanation of the `max_buf` subsystem attribute, see the `sys_attrs_scfs_client(5)` reference page.

4.3.4 Monitoring SCFS Activity

The activity of the `scfs` kernel subsystem, which implements the SCFS I/O serving and data transfer capabilities, can be monitored by using the `scfs_xfer_stats` command. You can use this command to determine what SCFS file systems a node is using, and report the SCFS

usage statistics for the node as a whole, or for the individual file systems, in summary format or in full detail. This information can be reported for a node as an SCFS server, as an SCFS client, or both.

For details on how to use this command, see the `scfs_xfer_stats(8)` reference page.

4.4 SCFS Failover

The information in this section is organized as follows:

- SCFS Failover in the File Server Domain (see Section 4.4.1 on page 4–8)
- Failover on an SCFS Importing Node (see Section 4.4.2 on page 4–8)

4.4.1 SCFS Failover in the File Server Domain

SCFS will failover if a node fails in the FS domain because the file systems are CFS and/or AdvFS.

4.4.2 Failover on an SCFS Importing Node

Failover on an SCFS importing node relies on NFS cluster failover. As NFS cluster failover does not exist on Tru64 UNIX, and there are no plans to implement this functionality on Tru64 UNIX, there are no plans to support SCFS failover in a compute domain.

HP AlphaServer SC uses an automated mechanism to allow `pfsmgr/scfsmgr` to unmount PFS/SCFS and remount when the importing SCFS node fails. The automated mechanism unmounts the file systems and remounts the file systems when the importing node reboots.

Note:

This implementation does not imply failover.

4.4.2.1 Recovering from Failure of an SCFS Importing Node

Note:

If the automated mechanism fails, a cluster reboot should not be required to recover. It should be sufficient to reboot the SCFS importing node.

The automated mechanism runs the `scfsmgr sync` command on system reboot. There are two possible reasons why the `scfsmgr sync` command did not remount the file systems:

- A problem in `scfsmgr` itself. Review the log files below for further information:
  - The event log (by using the `scevent` command), and look in particular at SCFS, NFS, and PFS classes.
  - The log files in `/var/sra/adm/log/scmountd`. Review the log file on the domain where the failure occurred and not on the management server.
  - The `/var/sra/adm/log/scmountd/scmountd.log` file on the management server. This log file may contain no direct evidence of the problem. However, if after member 1 failed, `srad` failed to failover to member 2, the log file reports that the domain did not respond.
- The file system was not unmounted by Tru64 UNIX, even though the original importing member has left the cluster.

Note:

If this occurs, the `mount` or `unmount` commands might hang and this will not be reflected in the log files. In the event of such a failure, send log files and supporting data to the HP AlphaServer SC Support Centre for analysis and debugging.

To facilitate analysis and debugging, follow these steps:

1. To gather information on why the file system was not unmounted, run `dumpsys` from all nodes in the domain. Send the data gathered to the local HP AlphaServer SC Support Center for analysis.
2. Check if any users of the file system are left on the domain.
3. Run the `fuser` command on each node of the domain and kill any processes in that area using the file system.
4. If you are using PFS on top of SCFS, run the `fuser` command on the PFS file system first, and then kill all processes using the PFS file system.
5. Unmount the PFS file system using the following command (assuming domain name `atlasD2` and PFS file system `/pdata`):
   ```
   # scrun -d atlasD2 -m all /usr/sbin/umount_pfs /pdata
   ```
   The `umount_pfs` command may report errors if some components have already mounted cleanly.

   Check whether the unmount occurred using the following command:
   ```
   # scrun -d atlasD2 -m all "/usr/sbin/mount | grep /pdata"
   ```

Note:

If still mounted on any node, repeat the `umount_pfs` command on that node.

6. Run the `fuser` command on the SCFS file systems and kill all processes using the SCFS.
7. Unmount the SCFS using the following command (where `/pd1` is an SCFS):
   `# scrun -d atlasD2 /usr/sbin/umount /pd1`
8. Once the SCFS has been unmounted, remount the SCFS file system using the following command:
   `# scfsmgr sync`

Note:

Step 7 and 8 may fail either because one or more processes could not be killed or because the SCFS still cannot be unmounted. If that happens, the only remaining option is to re-boot the cluster. Send the `dumpsys` output to the local HP AlphaServer SC Support Center for analysis.

5
Recommended File System Layout

The information in this chapter is arranged as follows:

- Recommended File System Layout (see Section 5.1 on page 5–2)

5.1 Recommended File System Layout

Before storage and file systems are configured, the primary use of the file systems should be identified.

PFS and SCFS file systems are designed and optimized for applications that need to dump large amounts of data in a short period of time and should be considered for the following:

- Checkpoint and restart applications
- Applications that write large amounts of data

Note:

The HP AlphaServer SC Interconnect reaches its optimum bandwidth at message sizes of 64KB and above. Because of this, optimal SCFS performance will be attained by applications performing transfers that are in excess of this figure. An application performing a single 8MB write is just as efficient as an application performing eight 1MB writes or sixty-four 128KB writes — in fact, a single 8MB write is slightly more efficient, due to the decreased number of system calls.

Example 5–1 below displays sample I/O block sizes. To display sample block sizes, run the Tru64 UNIX `dd` command.

Example 5–1 Sample I/O Blocks

```
time dd if=/dev/zero of=/fs/hsv/fs0/testfile bs=4k count=102400
102400+0 records in
102400+0 records out

real   68.5
user   0.1
sys    15.4
time dd if=/dev/zero of=/fs/hsv/fs0/testfile bs=1024k count=400
400+0 records in
400+0 records out

real   8.3
user   0.0
sys    1.8

atlas64 #
```

PFS and SCFS file systems are not recommended for the following:

- Applications that only access small amounts of data in a single I/O operation:

  - PFS/SCFS is not recommended for applications that only access small amounts of data in a single I/O operation (for example, 1KB reads or writes are very inefficient).
  - PFS/SCFS works best when each I/O operation has a large granularity (for example, a large multiple of 128KB).
  - With PFS/SCFS, if an application is writing out a large data structure, (for example, an array), it would be better to specify to write the whole array as a single operation, than to write it as one operation per row or column. If that is not possible, then it is still much better to access the array one row or column at a time than to access it one element at a time.

- Applications that require caching of data
- Serial/general workloads:
  - PFS and SCFS file systems are not suited to serial/general workloads due to limitations in PFS `mmap` support and lack of `mmap` support when using SCFS on CS domains. Serial/general workloads can use linkers, performance analysis, and/or instrumentation tools which require use of `mmap`.
  - Some of the limitations of PFS and SCFS can be overcome if the PFS is configured with a default stripe width of one. Some of the limitations can also be overcome if the serial workloads are run on the FS domains on nodes which do not serve the file system. For example, if the FS domain consists of 6 nodes, and 4 of these nodes were the `cfsmgr` for the component file systems for PFS, by running on one of the other two nodes you should be able to see a benefit for small I/O and serial/general workloads.
  - If the workload is run on nodes that serve the file system, the interaction with remote I/O and the local jobs will be significant.

These applications should consider an alternative type of file system.

Note:

Alternative file systems that can be used are either locally available file systems, or Network File Systems (NFS).

To configure PFS and SCFS file systems in an optimal way, the following should be considered:

1. Stride Size of the PFS
2. Stripe Count of the PFS
3. Mount Mode of SCFS

5.1.1 Stride Size of the PFS

The stride size of PFS should be large enough to allow the double buffering effects of SCFS operations to take place on `write` operations. The minimum recommended stride size is 512K. Depending on the most common application use, the stride size can be made larger to optimize performance for the majority of use. This will depend on the application load in question.

5.1.2 Stripe Count of the PFS

The benefits of a larger stripe count are to be seen where multiple writers are all writing to just one file. Performance improvements are also noticeable, however, where multiple processes are all writing to multiple files. This will depend on the most common application type used.

As the stripe count of the PFS is increased, the penalty applied to operations, such as `getattr` which access each component that the PFS file is striped over, will also increase. You are not advised to stripe the PFS for more than eight components, especially if there are significant meta data operations on the specific file system.

If there are operations that require `mmap()` support, the recommended configuration is a stripe count of one (for more information, see the *HP AlphaServer SC Administration Guide and Release Notes*).

Note:

Having a stripe count of one does not mean that the number of components in the PFS is one. It means that any file in the PFS will only use one component to store data.

5.1.3 Mount Mode of the SCFS

In general, the FAST mode for SCFS is configured. This allows a fast mode operation for reading and writing data, however, there are some caveats with this mode of operation:

- UBC is not used on the client systems (so in general `mmap` operations will fail). To disable SCFS/FAST mode, and enable SCFS/UBC mode on a SCFS/FAST mounted file system, set the `execute` bit on a file.

Note:

On a typical file system, the best performance will be obtained by writing data in the largest possible chunks. In all cases, if the files are created with the `execute` bit set, then the characteristics will be that of NFS on CS domains, and AdvFS on FS domains. In particular, for small writers or readers that require caching it is useful to set the `execute` bit on files.

- Small data writes are slow due to the direct communication between the client and server and the additional latency that this entails.
- If a process or application requires read caching, this is not available since each read request will be directed to the server.

Note:

If any of the above characteristics are an important consideration, then the SCFS should be configured in UBC mode. SCFS in UBC mode offers exactly the same performance characteristics as NFS. If SCFS/UBC is to be considered, then one should review why NFS was not configured originally.

5.1.4 Home File Systems and Data File Systems

With home file systems, you should configure the system to use NFS due to the nature and type of usage.

Note:

SCFS/UBC configured file systems, which are equivalent to NFS, can also be considered if the home file system is served by another cluster in the HP AlphaServer SC system.

File systems that are used for data storage from application output, or for checkpoint/restart, will benefit from an SCFS/PFS file system.

For more information on NFS, refer to the *Compaq TruCluster Server Cluster Technical Overview*. For information on configuring NFS, refer to the *Compaq TruCluster Server Cluster Administration Guide.*

For sites that have a single file system for both home and data files, it is recommended to set the `execute` bit on files that are small and require caching, and use a stripe count of 1.

6

Streamlining Application I/O Performance

The file system for the HP AlphaServer SC system and individual files can be tuned for better I/O performance. The information in this chapter is arranged as follows:

- PFS Performance Tuning (see Section 6.1 on page 6–1)
- FORTRAN (see Section 6.2 on page 6–4)
- C (see Section 6.3 on page 6–5)
- Third Party Applications (see Section 6.4 on page 6–5)

6.1 PFS Performance Tuning

PFS specific `ioctls` can be used to set the size of a stride and the number of stripes in a file. This is normally done just after the file has been created and before any data has been written to the file, otherwise the file will be truncated.

The default stripe count and stride can be set in a similar manner.

Example 6–1 below describes the code to set the default stripe count of a PFS to the value input to the program. Similar use of `ioctls` can be incorporated into C code or in FORTRAN *via* a callout to a C function.

A FORTRAN unit number can be converted to a C file descriptor *via* the `getfd(3f)` function call (see Example 6–2 and Example 6–3).

Example 6–1 Set the Default Stripe Count of a PFS to an Input Value

```
#include <stdio.h>
#include <fcntl.h>
#include <inttypes.h>
#include <libgen.h>
#include <string.h>
#include <sys/fs/pfs/common.h>
#include <sys/fs/pfs/map.h>

static char *cmd_name = "pfs_set_stripes";
```

```
static int def_stripes = 1;
static int max_stripes = 256;

void
usage(int status, char *msg)
{
    if (msg)
    {
        fprintf(stderr, "%s: %s\n", cmd_name, msg);
    }

    printf("Usage: %s <filename> [<stripes>]\nwhere\n\t<stripes> defaults to
%d\n", cmd_name, def_stripes);

    exit(status);
}

int
main(int argc, char *argv[])
{
    int fd,status,stripes=def_stripes;
    pfsmap_t map;

    cmd_name = strdup(basename(argv[0]));

    if (argc < 2)
    {
        usage(1, NULL);
    }

    if ((argc == 3) && (((stripes=atoi(argv[2])) <= 0) || (stripes >
max_stripes)))
    {
        usage(1, "Invalid stripe count");
    }

    if ((fd = open(argv[1], O_CREAT | O_TRUNC, 0666)) < 0)
    {
        fprintf(stderr,"Error opening file %s \n",argv[1]);
        exit(1);
   }

    /*
    * Get the current map
    */

    status = ioctl(fd, PFSIO_GETDFLTMAP, &map);
    if (status != 0)
    {
        fprintf(stderr,"Error getting the pfs map data \n");
        exit(1);
    }
```

```
    map.pfsmap_slice.ps_count = stripes;

    status = ioctl(fd, PFSIO_SETDFLTMAP, &map);

    if (status != 0)
    {
        fprintf(stderr,"Error setting the pfs map data \n");
        exit(1);
    }

    exit(0);
}
```

Example 6–2 and Example 6–3 describe code samples for the `getfd(3f)` function call.

Example 6–2 Code Samples for the getfd Function Call

```
IMPLICIT NONE

      CHARACTER*256 FILEN
      INTEGER ISTAT

      FILEN = './testfile'

      OPEN (
     :        FILE = FILEN,
     :        FORM = 'UNFORMATTED',
     :        IOSTAT = ISTAT,
     :        STATUS = 'UNKNOWN',
     :        UNIT = 9
     :        )

      IF (ISTAT .NE. 0) THEN
          WRITE (*,155) FILEN
          STOP
      ENDIF
      CALL SETMYWIDTH(9, 1, ISTAT) ! Ths will truncate the file and set pfs width
to 1

      IF (ISTAT .NE. 0) THEN
          WRITE (*,156) FILEN
          STOP
      ENDIF


155   FORMAT ('Unable to OPEN file ',A)
156   FORMAT ('Unable to set pfs width on file ',A)

      END
```

Example 6–3 Code Samples for the getfd Function Call

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <inttypes.h>
#include <sys/fs/pfs/common.h>
#include <sys/fs/pfs/map.h>

int getfd_(int *logical_unit_number);

void setmywidth_(int *logical_unit_number,int *width, int *error)
{
    pfsmap_t map;
    int fd;
    int status;

    fd = getfd_(logical_unit_number);

    status = ioctl(fd, PFSIO_GETMAP, &map);
    if (status != 0)
    {
       *error = status;
       return;
    }

    map.pfsmap_slice.ps_count = *width;

    status = ioctl(fd, PFSIO_SETMAP, &map);
    if (status != 0)
    {
       *error = status;
       return;
    }

    *error = 0;
    return ;
}
```

6.2 FORTRAN

FORTRAN programs that write small records using, for example, formatted write statements will not perform well on an SCFS/FAST mounted PFS file system. To optimize performance of a FORTRAN program that writes in small chunks on an SCFS/FAST mounted PFS file system, it may be possible to compile the application with the option: `-assume buffered_io`.

This will enable buffering within FORTRAN so that data will be written at a later stage once the size of the FORTRAN buffer has been exceeded. In addition, for FORTRAN applications, the FORTRAN buffering can be controlled by an environment variable `FORT_BUFFERED`.

Individual files can also be opened with buffering set to `on` by explicitly adding the `BUFFERED` directive to the FORTRAN `open` call.

Note:

The benefit of using the option: `-assume buffered_io` is dependent on the nature of the applications I/O characteristics. This modification is most appropriate to applications that use FORTRAN formatted I/O.

6.3 C

If the Tru64 UNIX system `read()` and `write()` function calls are used, then the data is passed directly to the SCFS or PFS `read` and `write` functions.

However, if the `fwrite()` and `fread()stdio` functions are used, then buffering can take place within the application. The default buffer for `fwrite()` and `fread()` functions is set at 8K. This buffer size can be increased by supplying a user-defined buffer and using the `setbuffer()` function call.

Note:

There is no environment variable setting that can change this unless a special custom library is developed to provide the functionality. Buffering can only take place within the application for `stdio fread()` and `fwrite()` calls, and not `read()` and `write()` function calls.

For more information on the `setbuffer()` command, read the manpage.

6.4 Third Party Applications

Third Party Applications I/O may be improved by enabling buffering for FORTRAN (refer to Section 6.2), or by setting PFS parameters on files you know about that are not required to be created by the code.

Note:

Care should be exercised when setting the default behaviour to buffered I/O. The nature and interaction of the I/O has to be well understood before setting this parameter. If the application is written in C, there are no environment variables that can be set to change the behaviour.

Index

A

Abbreviations, xiii

C

CFS (Cluster File System)
  Overview, 1–3
CFS Domain
  Overview, 1–2
Cluster File System
  See CFS
Code Examples, xix
CS Domain, 1–3

D

Documentation
  Conventions, xviii
  Online, xix

E

Examples
  Code, xix
External Storage
  See Storage, Global

F

FAST Mode, 2–3
File System
  Recommended, 5–1
File System Overview, 2–1
FS Domain, 1–3, 4–2

I

Internal Storage
  See Storage, Local
Ioctl
  See PFS

L

Local Disks, 1–3

P

Parallel File System
  See PFS
PFS (Parallel File System), 1–5
  Attributes, 3–2
  Ioctl Calls, 3–9
  Optimizing, 3–7
  Overview, 2–4, 3–2
  Planning, 3–4
  Storage Capacity, 3–4
  Structure, 3–4
  Using, 3–6

R

RAID, 2–12

S

SCFS, 1–5, 2–2
    Configuration, 4–2
    Failover, 4–8
    Overview, 4–2
    Tuning, 4–5
Storage
    Global, 2–10
    Local, 2–9
    Overview, 2–1, 2–8
    System, 2–12
Stride, 3–3
Stripe, 3–3

U

UBC Mode, 4–3